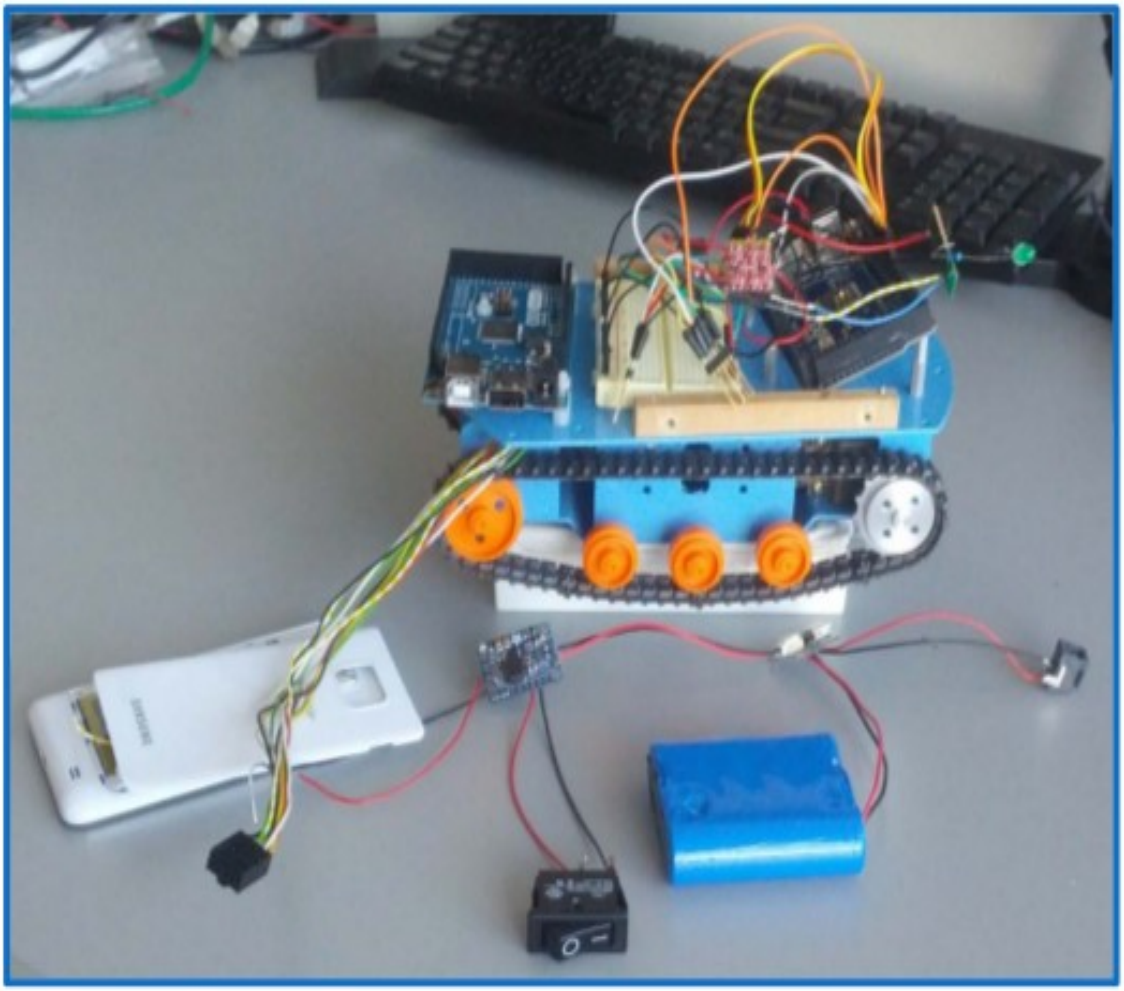
Programming a line-follow robot

Embedded Software - Robot Lab



Group Number: 25
Student 1:
Thijs Boumans, 4214854
Student 2:
Stephan Neevel, 4232623
Date: January 26th, 2015

Table of Contents

Introduction

The report is about programming a line-follow robot. The setup of the robot is the following: a smartphone is placed on the robot. This smartphone uses its camera to record the line. These images are sent to the laptop via wifi and are then processed to detect a line. The laptop sends directions to the Arduino in robot via bluetooth and the robot will drive and follow the line based on these commands. There is also an ultrasonic sensor facing to the front of the robot. When it detects something, the robot should stop in time not to bump into it.

The report will cover the requirements of the programming, testing whether the H-bridge is active low or high and the used timer settings on the Arduino. It will also cover the message structure of the so called Twist message and how the motors react to this. Next comes the line detection algorithm and how the Twist messages are constructed based on this.

Requirements

To develop a clear basis for the programming of the robot, requirements have been drawn. These are divided in functional and non-functional requirements.

Functional requirements

-The robot can drive around based on geometry_msgs/Twist messages published on the topic cmd_vel.
-The robot will stop when an object of at least 4 cm width and 3 cm height is placed within 20 cm in front of it, so it doesn't collide with it. Due to the form of the tracks on the robot, we estimated that the robot can drive over objects that are a maximum of 3 cm high. Also, from our experience using the HC -SR04 ultrasonic sensor, we noted it can reliably 'see' objects of 4 cm width within 1 meter distance.
-The robot should be able to drive at different speeds based on the geometry_msgs/Twist messages published on the topic cmd_vel. It should remain able to meets its other requirements.
-The robot is able to follow the line in 4 different test tracks. One of the tracks contains lines crossing each other, and the robot should be able to cope with this. The line can be as little as 0.4 cm wide, Furthermore, the sum of the difference between the background and the line of all three RGB channels could be as little as 90. A bit of testing with different RGB values and experience with other line following robots, we came to this number.
-The timers in the Arduino sketch are set via the AVR timing registers.
-The speed of the robot is adjusted based on the distance to the robot in front when several robots drive on the same test track. The minimal distance to the robot in front should always be 20 cm. We estimate this is a safe distance to avoid collision.
-The robot must stop when no new commands have been given to the Arduino from the laptop for a time of 2 seconds. This is to avoid the robot from going rogue.
-The ROS line-follow node should be independent of the implementation of the robot; it just uses the geometry_msgs/Twist messages and publishes it on the topic cmd_vel.

Non-functional
-The line follow setup will consist of an android smartphone, a robot with Arduino, and a laptop working together.
-The software should be independent of laptop choice and android smartphone choice.
-The roundtrip delay from the smartphone, to the laptop, to the Arduino can be as high as 100 ms. We estimated that with a high disturbance, and a short distance of the wifi and bluetooth, this is a reasonable number.

Testing the H-bridge

In order to find out if the H-bridge was active high or active low triggered we ran the following code:

```
#define REV1 7
#define EN1 24
#define FWD1 6
#define REV2 3
#define EN2 25
#define FWD2 2
#define fuel 1
void setup(){
        pinMode(FWD1 ,OUTPUT);
        pinMode(FWD2 ,OUTPUT);
        pinMode(EN1 ,OUTPUT);
        pinMode(EN2 ,OUTPUT);
        pinMode(REV1 ,OUTPUT);
        pinMode(REV2 ,OUTPUT);
        pinMode(trigPin, OUTPUT);
        pinMode(echoPin, INPUT);
       digitalWrite(REV2,LOW);
       analogWrite(FWD2,255);
       digitalWrite(EN2,HIGH);
}
```

This activated the motors and therefore showed us that the H-bridge is active high.

Used timer settings on the Arduino

We used AVR timing registers which were activated by the following code:

```
void setupTimeoutInterrupt(){
        noInterrupts();
        TIMSK1=0x01; // enabled global and timer overflow interrupt;
        TCCR1A = 0x00; // normal operation page 148 (mode0);
        TCNT1=0x0000; // 16 bit counter register
        TCCR1B = 0x04; // start timer/ set clock increase for longer timeout settings
        interrupts();
}
```

Upon triggering the interrupt the following code was run:

```
 ISR(TIMER1_OVF_vect) {
        timesteps ++;
        if(timesteps>2){
                leftTrack=0;//disable engine in next loop
        rightTrack=0;
  }
}
```

We can reset the timer by using the following function. It disables interrupts to make sure there is no shared data problem with the ISR.

```
void resettimer(){
  noInterrupts();
  timesteps =0;
  interrupts();
}
```

Twist-message to motor-interface mapping

Interfacing between the ROS messages and the motor happens via the driveCallback. This callback sets the engine power using the following code:

```
void driveCallback(const geometry_msgs::Twist &msg){
  resettimer();//reset timer as we are still connected to the laptop
  float x = msg.linear.x;
  float z = msg.angular.z;

    leftTrack=x-z;  //power settings
    rightTrack=x+z;
}
```

After that turning on a motor, in this case the right motor, is as easy as

```
    digitalWrite(REV2,LOW);
    analogWrite(FWD2,power*rightTrack);
    digitalWrite(EN2,HIGH);
```

The power variable is a constant that we use in order to control the speed of the robot, setting it to 255 means the robot moves at full speed whereas on speed 128 it only moves at half of it's maximum speed (or rather with only half of it's maximum power). The angular z variable is positive when the robot has to turn to the left.

The line detection algorithm and generating Twist messages

When the picture of the line arrives on the computer, the algorithm processes it and detects the line and send a Twist message back to the robot. The algorithm first starts by scaling, rotating and warping the image. The picture arrives on its side, so to make viewing possible it is rotated in the right direction. To

improve the speed of the algorithm, the picture is scaled down by 90 percent, to a picture of 128*72 pixels. The smartphone is placed at an angle in the robot, so lines that are parallel on the track will not be parallel on the image. To adjust for this and get the lines parallel again, the picture is warped. This is necessary because angles calculated from this warped picture will correspond one on one with the angles on the track. At least that was what we planned because in the end when the line following was graded, the smartphone was taped horizontally to the robot, so warping the picture wasn't needed.

The next step in the algorithm used during the grading is converting the picture into gray scale. A threshold of the darkness of the line to be followed had to be manually determined. All pixels that have a value below (darker than) the value of threshold are then turned black, all that have a higher value (lighter) are turned white, removing noise. So this leaves a black line and a white background. The following step is scaling the picture down to a picture of 1 by 3 pixels. This scaling linearly interpolates between the pixels. When the line goes to the left on the track, there are more black pixels on the left of the screen, so the left pixel will turn black in this final picture, and the others white. The same goes for other directions of the line. Based on this final picture a Twist message is generated. When the left pixel is black, linear.x is 1, and angular.z is 1. When the right pixel is black, linear.x is 1 and angular.z is -1. When the middle pixel is black, the robot has to go straight and just linear.x is 1. The robot will execute the last sent command until a different command has been sent to it, (or there is a timeout in the communication).

Test results using the algorithm

During the grading for following the line, the algorithm proved working moderately. The robot followed the straight line and about 3/4 of the curved track, but with a lot of zigzagging. This can be explained by the fact that the line is relatively small in width compared to the whole picture and general latency of the system. The algorithm interpolates the final picture with the left or right pixel to be black. So the robot will often get a late update that it has to turn. Another test result was that the robot failed to follow a line that wasn't as dark as the black line, this can be explained by fact that the threshold should be different when there is a different color line.

A better line following algorithm

To improve the line following we would probably have to implement a different algorithm. This new algorithm could first start by scaling down, rotating and (if needed) warping the picture. The next step could be using a Gaussian filter to filter out noise in the picture. Following that could be a canny edge detection algorithm, resulting in a picture with a black background and two white lines that are the edges of the line to be followed. Continuing with a Hough line detection algorithm, the location of these edges on the picture can be known, as this function returns vectors of these lines. With these vectors, the angle between the edge lines detected and the vertical centerline of the picture can be calculated. To decide which line to follow we could follow the longest detect line that is not part of the edge of the picture itself. If this angle is big, a twist message can be sent to turn with a high angular velocity and a low linear velocity, if it is small with a low angular velocity and higher linear velocity. When the angle is close to zero, the message would be a medium linear velocity. This new algorithm doesn't involve a threshold that has to be manually set, so the problems arising from this would be gone. It would also give less zigzagging, because it would be more proactive in determining the correct direction.

We have tried multiple algorithms and variations, but could not quite get them to work that well, so for the grading we finally decided to stick with the algorithm as described in the first section, because it was the one that managed to get most of the test track.